The RAGE Game Software Components Repository Supporting Applied Game Development

A. Georgiev[1], A. Grigorov[1,6], B. Bontchev[1], P. Boytchev[1], K. Stefanov[1],
W. Westera[2], E. Nyamsuren[2], K. Bahreini[2], R. Prada[3], P. Hollins[4], P. Moreno[5]
[1] corresponding *Sofia University "St. Kliment Ohridski", Faculty of Mathematics and Informatics, Bulgaria, krassen@fmi.uni-sofia.bg, {atanas,alexander.grigorov,bbontchev,boytchev,krassen}@fmi.uni-sofia.bg*
[2] *Open University of the Netherlands {wim.westera,enkhbold.nyamsuren,kiavash.bahreini}@ou.nl*
[3] *University of Lisbon, Portugal rui.prada@tecnico.ulisboa.pt*
[4] *The University of Bolton, UK pah1@bolton.ac.uk*
[5] *Universidad Complutense de Madrid, Spain pablom@fdi.ucm.es*
[6] *Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Bulgaria grigorov@math.bas.bg*

Abstract

This paper presents the architecture of the RAGE repository, a unique and dedicated infrastructure that provides access to a wide variety of advanced technology components for applied game development. The RAGE project, which is the principal Horizon2020 research and innovation project in applied gaming, will develop software components (RAGE software assets) that are reusable across a variety of game engines, game platforms and programming languages. The RAGE repository provides storage space for these assets and associated artefacts and is designed as an asset life-cycle management system for defining, publishing, updating, searching and packaging for distribution of the assets. The repository will be embedded in a social platform for asset developers and other stakeholders. A dedicated Asset Repository Manager provides the main functionality of the repository and its integration with other systems. Additional Tools supporting the Asset Manager are also presented and discussed. When the RAGE repository is fully operational, applied game developers will be able to enhance the quality of their games through the application of selected advanced game software assets. By making available the RAGE repository system and software assets the RAGE project's aim is to stimulate the development and uptake of the Applied Games Industry IN Europe.

Keywords: *software assets, serious games, asset repository, taxonomy tools, metadata editor, reuse.*

1. Introduction

Applied gaming is highlighted as one of the main priorities in Horizon2020, the Research and Innovation Programme of the European Commission. Policy makers of the European Commission envision a flourishing applied games industry that helps to address a variety of societal challenges in education, health, social cohesion and citizenship, and equally one that stimulates the creation of jobs in the creative industry sector.

Although applied or serious games have been successfully employed in education and training settings across a wide and varied range of application domains, seizing the full potential of applied games has been challenging. In contrast, the leisure games industry is an established industry dominated by large international hardware vendors (e.g. Sony, Microsoft and Nintendo) and large publishers and retailers. Conversely, the applied game industry is fragmented across a plethora of small independent businesses with limited interconnectedness and knowledge exchange [1-2].

The RAGE project [3] aims to stimulate the applied game industry by making available a set of advanced reusable game technology components (software assets) that game studios can easily integrate in their game development projects. Applied game studios would benefit from using state-of-the-art technologies, while incorporating complex pedagogic technical functionality would become easier and quicker, and the cost of development would be reduced. The software assets cover a variety of functionalities including game analytics, emotion recognition, assessment, personalised learning, game balancing and player-centric adaptation, procedural animation, language technologies, interactive storytelling, and social gamification.

While the main research goal of the RAGE project is to support the applied game industry through making available a large set of reusable, advanced software components (applied gaming assets), this paper focuses on the design of the repository infrastructure that is required to support the processes of development, reuse and sharing of applied gaming assets. This paper presents the asset repository architecture and the associated asset development methodology. We first present the related work efforts, discuss our approach (research method), describe the software asset concept, provide details of the design and implementation of the back-end repository system architecture and corresponding front-end tools, and we conclude with a brief description of our initial experiments with the infrastructure, analysis and identification of further development and research efforts.

2. Related work

Asset-based software development relies on reusing well documented and cohesive software artefacts and, therefore, it is inconceivable without a platform for storing and accessing assets. An asset repository as a software tool is defined by Ackerman and colleagues [4] for storing and retrieving reusable assets and managing asset access control for asset producers and consumers, according to the phases of the asset life cycle. They introduce the IBM Rational Asset Manager (RAS) repository, which handles tasks and activities of software asset producer, consumer and subscriber roles, while offering reduced production costs and improved software quality. In order to facilitate cross-project reuse of assets, the RAS model provides monitoring of asset categorization and usage together with multi-platform compliance management. RAS is currently a standard supported by Open Management Group (OMG).

Another example for a RAS-based asset repository is the Atego Asset Library [5], which is a scalable Web-based repository for reusable software engineering artefacts. It is based on OMG RAS and integrates Unified Modelling Language (UML) and Systems Modelling Language (SysML) in order to facilitate asset reuse at design time. Currently, the tool is supported as PTC Integrity Asset Library[1] and, besides the publishing, finding and reuse of assets, provides services as interest registry and notification, automatic file interrogation, traceable links and reuse metric dashboard.

Extensions of the OMG RAS have been proposed for designing open source Web-based asset repositories providing advanced classification, search and utilization of reusable software assets of various types. The OpenCom asset repository was created as a supporting tool of Shanghai Component Library [6] based on an extension of OMG RAS profile aiming at collaborative creation of knowledge by web users. The Lavoi free source asset repository [7] was developed based on an extension of the component profile of OMG RAS broadening the categories about classification, solution, usage and related assets.

In the games domain various existing platforms are promoting reusability in both serious games and leisure games. The Unity Asset Store (https://www.assetstore.unity3d.com/) is an example of a successful online market place for game objects. Most of the objects are media objects (e.g. terrains, audio, buildings, weapons), but an increasing number of software modules (e.g. analytics, cloud backend, game AI) are becoming available. Unfortunately, most of the software objects can only be reused in the Unity game engine. Various other online platforms offer reusable game objects, for instance the Guru asset store (https://en.tgcstore.net/), Game Salads (http://gshelper.com/),

[1] http://www.ptc.com/model-based-systems-engineering/integrity-modeler/asset-library

GameDev Market (https://www.gamedevmarket.net/), Unreal Marketplace (https://www.unrealengine.com/marketplace), and Construct 2 (https://www.scirra.com/store), but similarly their main focus is on user-interface objects and templates, while scarce software components will only run on one preferred game engine. The Intel® XDK HTML5 Cross-platform Development Tool [8] offers an asset manager for game development in conjunction with several game platforms, be it that its focus is not on software assets, but on media assets, that is, audio-visual game objects to be included in a project. In RAGE the focus is on software assets, reusable components adding specific (pedagogic) functionality for applied game development.

A similar attempt related to using a digital repository of metadata resources for education, combined with a portal for the respective community of practices build around the repository, is described in [9]. Other approaches to endowing digital libraries with adaptability capabilities in order to scaffold and enhance end user experience are presented in [10]. Similar attempts inside the GALA Network of Excellence are the Service-Oriented Architecture (SOA) framework for applied games [11] and the repository for exchange of game resources [12].

Although RAS is a complete architecture, it was not immediately applicable to the RAGE project. The main distinctions being: (1) RAS is too general and some of its elements were not suitable for RAGE assets, additionally, RAGE required data which were not present in RAS; (2) RAGE used a different interpretation of some terms, and this might lead to confusion if other standards were being adopted as-they-are. For example, within RAGE, the asset is strictly a software asset with some predefined structure, while in other models assets could be an image or a voice recording.

The RAGE project, however, is unique at proposing a specific software architecture. The idea about the use of the metadata schema as an input for generating the metadata editor, the configuration editor and the validator, is unique [16]. Additionally the metadata schema is used in order to index all asset metadata inside the repository. This means that changing the metadata schema will not affect any one of the software programs that constitute the repository, only their interface part defined by the schema.

The RAGE component-based software architecture preserves the portability of assets and that supports data interoperability between the assets [13]. The concept underpinning interoperable software components as reusable game elements with specific objective and functionality is also unique. The approach captures more detail than RAS architecture and is more focused and related to the domain of applied games. This results in a unique architecture, more detailed, but conversely simpler and easier to be deploy in comparison to RAS architecture. This results in a more efficient and user friendly architecture.

Moreover, it transcends the SOA framework by allowing client-side assets that are directly integrated in the game engine' Thereby manifest limitations of SOA can be bypassed, such as the requirement of constant online connectivity, diminished flexibility such as customisation and configuration of services by service consumers, reduced system performance due to additional overheads associated with SOA and network calls. In contrast with software components that are available for proprietary game engines, RAGE-compliant components are interoperable and designed for porting to multiple game engines and platforms. The usage of the Asset manager in run time is also unique and provides more flexibility and efficiency for implementing reusable game assets in real game settings.

3. RAGE Software Assets

A RAGE asset includes a self-contained software component related to computer games, intended to be reused and or repurposed across different game platforms. Its formal definition is compliant with the asset definition of the W3C ADMS Working Group [14], which refers to abstract entities that reflect some "intellectual content independent of their physical embodiments". In principle, not all assets are required to include software, however this paper focusses on software assets. In addition to the software component, the RAGE asset includes value-adding services and attributes that facilitate their use, e.g. instructions, tutorials, examples and best practices, empirical research papers that validate their usage, instructional design guidelines, connectors to major game development platforms, test plans, test scripts, design documents, data capacity, and content authoring tools/widgets for game content creation.

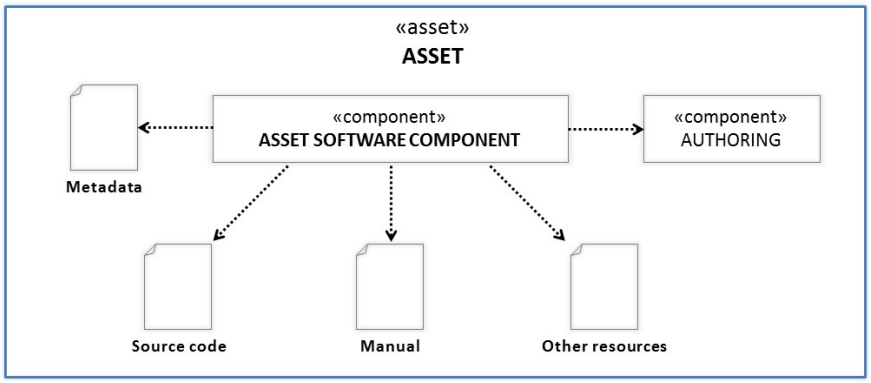

Figure 1. Conceptual layout of a RAGE Asset

Figure 1 presents the general layout of a RAGE asset. It includes a software component that should comply with the RAGE component architecture, already described and validated in [15]. The RAGE architecture addresses both the internal workings of an asset and the level of interaction of assets with the outside world, including the mutual communications between assets.

For preserving the portability requirements the RAGE architecture avoids dependencies on external software frameworks and minimises code that may hinder integration with game engines. It relies on a limited set of standard software patterns and well-established coding practices. These include the "Publish/Subscribe" pattern, the "Singleton" pattern, the "Bridge" pattern, asset method calls and web services. Nevertheless, portability across different programming languages is sometimes affected by the different natures and execution modes of the supported languages, for instance interpreted languages such as JavaScript versus compiled languages as C++, C# and Java. Multi-threading versus single threading could hamper code portability. Still, the RAGE architecture was successfully validated for the languages mentioned above, be it that some minor syntax issues should be considered (e.g. the use of directory separators "/" versus "\" in Windows and Linux OS, respectively) and some IDE version issues (e.g. Visual Studio). A list of compatibility issues and their solutions have been reported in [13].

In addition, each RAGE asset contains metadata, which describe its content and functionality in a machine-readable format. This metadata is essential for being able to classify, store, organise, modify and search and retrieve assets from the repository. The metadata includes information that is critical for running the asset software in an operational environment, e.g. on a game platform. This relates to version information and information about dependencies on other software assets. Furthermore, the metadata provides information about intellectual property rights and the asset's potential usage within three distinct dimensions – technological, applied gaming and pedagogical. The RAGE metadata model is designed for defining the asset's metadata in all three dimensions, and to enable the proper implementation of the RAGE Asset repository system architecture [16].

4. Our approach

The research methodology for this study is based on the Rapid Application Development model [17]. We performed an extensive needs assessment study (partially described in [18]), including asset developers, educators and game producers. We have identified the services to be supported through the repository and other related tools and, in parallel, designed the RAGE metadata model to fit the specified domain of reusable gaming components (RAGE software assets). It was clear that we could not reuse any existing solution, but needed to design and implement our own software repository, targeting the identified needs and characteristics of the applied game domain.

We then provided the initial design of the RAGE asset which consisted of a software component additional artefacts, and the architecture of the RAGE software repository. The designs were aimed at supporting the development, storage, sharing and reuse of assets. At a later stage we provided details of the technical implementation of the software repository. We performed several iterations between these two stages until a stable and near complete solution was achieved. In the final stage we analysed the first use case scenarios of the repository through several client tools, arranged first evaluations of the repository, and collected ideas for its improvement in the next cycle.

The results of each stage of development are herein presented.

5. The Asset repository system architecture

Metadata is a key part of the information infrastructure necessary to help create order and provide a solid foundation for a number of information services such as descriptions, classifications, organizations, store, search, creation, modification and aggregation of information [19]. Rather than merely a software archive, the asset repository should be viewed as a system for managing the lifecycle of an asset. In the repository the asset's artefacts are collected and conceptually tied together by defining the metadata and the reference to underlying artefacts. In addition, the repository provides for the publication, updating, packaging for distribution and quality assurance, while accommodating different end-user tools.

Thereby the RAGE asset software repository is at the core of the asset development infrastructure. It is used to store and manage access to: (1) reusable game assets, (2) artefacts (resources within game assets), (3) metadata for game assets and artefacts, and (4) relationships between assets – dependencies, related assets, etc.

The Asset software repository leverages the discovery, development reuse and repurpose of game assets and artefacts. Facilitating game asset developers and consumers in all activities relating to the game asset lifecycle.

The main functions of the RAGE Asset software repository are as follows:

- Searching, finding and browsing software assets/artefacts
- Creating, updating, publishing, deleting and downloading assets/artefacts
- Versioning support, source code import from GitHub and integration with IDEs
- Harvesting of external repositories for game assets and metadata using the Open Archives Initiative - Protocol for Metadata Harvesting (OAI-PMH)
- Reviewing and rating assets/artefacts

To implement these functions, we designed the asset repository infrastructure in three tiers: client, service and data store tiers, which is visualised in Figure 2.

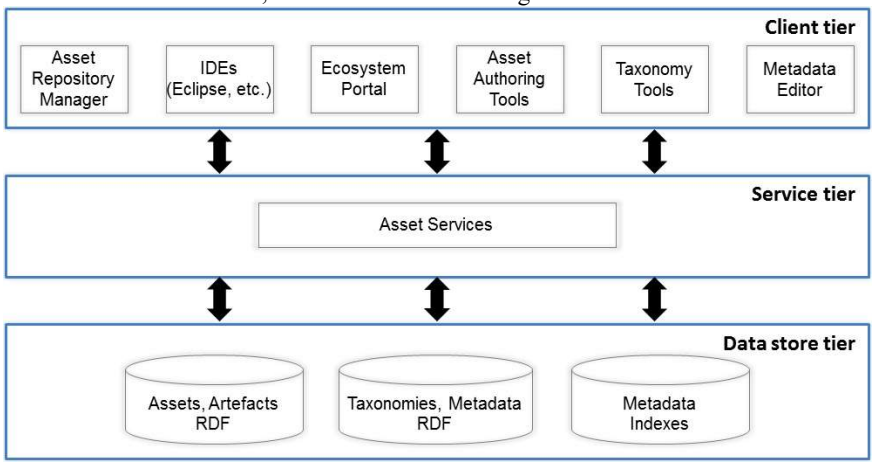

Figure 2. Asset Repository Architecture

6. Implementation of the asset repository system architecture

The next stage is the implementation of the Asset repository. Fedora [20] is used for storing assets, metadata and artefacts; Sesame [21] for managing RDF data and supporting classification and entities; and Solr [22] for indexing and searching the repository. The data store tier consists of these three components and is used to store game assets, artefacts, metadata, taxonomies and indexes:

- **Fedora** stores the game assets, artefacts and metadata using RDF as primary data format. When the repository is updated by creating, modifying or deleting resources, it generates specific events so that the Fedora indexer copies RDF from the repository to an external triple store to keep it synchronized with the repository. Fedora is flexible, well established and it ensures scalability and durability (the complete repository can be rebuilt at any time).

- **Sesame** is an architecture for the efficient storage and expressive querying of large quantities of metadata in RDF and RDF Schema. This includes creating, parsing, storing, inferencing and querying over such data. Sesame RDF triple store contains metadata from Fedora and classification taxonomies/vocabularies.
- **Solr** is an open source platform optimized for searching. Its major features are full-text search, sophisticated faceted search, almost real-time indexing, dynamic clustering of data, etc. It is used for creating full text indexes on the RAGE metadata fields, as well as for realizing full text search and faceted search.

The service tier is used for access and preservation of the assets and artefacts. For the implementation of this tier, we developed the following services that provide access to the underlying data store tier:

- **Fedora Services**. Fedora provides a general RESTful HTTP API for accessing repository resources through HTTP methods. It supports OAI-PMH [23] requests on content and metadata in the repository.
- **Sesame Services**. Sesame offers a RESTful HTTP interface supporting the SPARQL Protocol for RDF. It is a superset of the SPARQL and supports communication for Update operations and the Graph Store HTTP Protocol [24].
- **Solr Services**. Apache Solr exposes Lucene's Java API as REST-like API's which can be called over HTTP. The RESTful endpoints allow CRUD style operations to be performed on the repository resources.

In addition, for the service tier to provide access to the client tier, we developed **Asset Services** (see Figure 3) for composition and execution of workflows over RAGE Game Assets.

The client tier includes web-based applications, plug-ins for integrated development environments, and software components from the RAGE ecosystem that uses the services supported by Asset Repository Infrastructure. It includes:

- **The Asset Repository Manager** – we developed a web-based application embodying main functionalities for lifecycle management of assets and artefacts.
- **IDE plug-ins** – we developed rich clients consuming services from the Asset Repository service tier, which thus allows developers to manage assets from within their integrated development environment (IDE).
- **Other software components from the RAGE ecosystem**, such as the Ecosystem Portal (EP), which harvests assets and metadata through an OAI-PMH service provider from Asset Repository Service tier. While the repository system reflects and supports the asset creation process undertaken by IT developers, the EP is the publication platform and social communication platform targeting applied game developers.

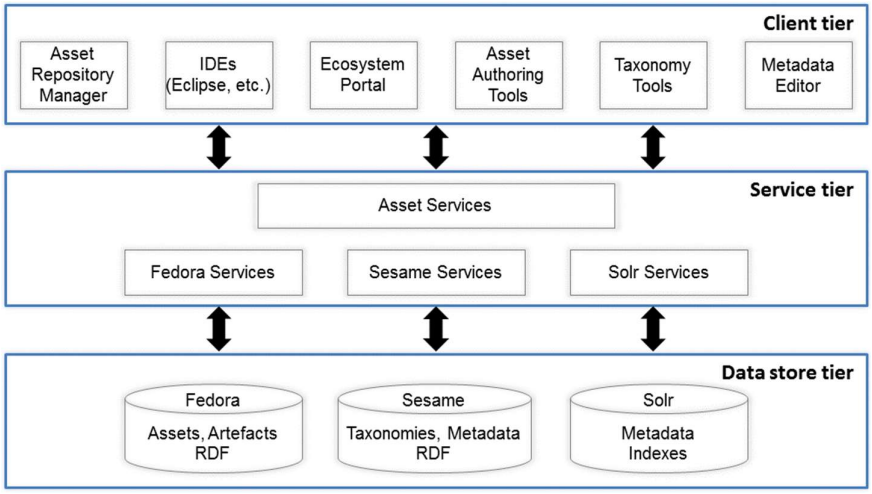

Figure 3. Asset Repository Architecture – Implementation

The Asset Repository services that we developed, constitute an open interface for creating, modifying, deleting, and searching RAGE assets. They are realised on top of REST APIs, JSON, JSON-LD [25], RDF and other widespread de facto standards. Based on the functionality exposed by these services, they can be grouped as:

- **Asset Access Services** defining an open interface for accessing assets within the RAGE Asset Repository allow for retrieving asset packages and metadata, and searching and browsing for assets using keywords and metadata fields. The search interface provides both full-text search and semantic search. Full-text search enables performing of natural language queries using keywords and phrases occurring in any of indexed asset's metadata elements. The semantic search is using SPARQL for querying on asset metadata and Simple Knowledge Organization System (SKOS) taxonomies data represented as RDF triples.
- **Asset Management Services** defining an open interface for administering assets, including creating, modifying, and deleting, provide an abstract level of the operations, thus hiding the complexities of the internal formats, protocols and procedures for storing an asset in the Asset Repository.
- **Taxonomy Services** defining an open interface for managing classification taxonomies and controlled vocabularies used in RAGE Asset Metadata Model [16] to classify and describe an asset in educational and gaming contexts. For representation and storing Asset Repository adopts SKOS standard [26].
- **Authentication and Authorization Services** provide access for organisational needs. These services are implemented on top of Fedora Authentication and Authorization framework [20].

7. Usage scenarios

To observe how the asset repository together with related client tools can support the asset developers and other users, and how effective and useful the services are, which it is offering, we designed various usage scenarios. Asset developers and game developers were involved in evaluating the functioning and usability of the repository.
In this section we present the scenarios.
To populate the repository with metadata we used four usage scenarios. The first scenario is publishing/updating a game asset through the web-based interface offered by the Asset Manager. The asset developer signs in, creates/selects an asset, enters/updates metadata and uploads artefacts or a packaged asset (see Figure 4).
The second scenario is publishing/updating a game asset from GitHub. The asset developer again should sign in the Asset Manager, creates/selects an asset, provides the GitHub repository identifier and credentials (if required). The files (artefacts) and metadata from GitHub are automatically harvested and published in the RAGE Asset Repository (using the GitHub API [27]). The user should also supply the rest of the required metadata.

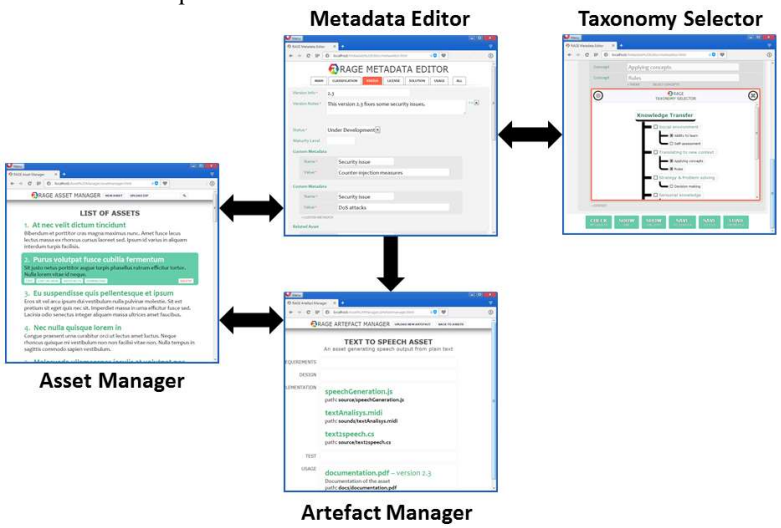

Figure 4. Using the RAGE Asset and Artefact managers, the RAGE Metadata editor and the RAGE Taxonomy selector to populate the repository

In the third scenario, we tested publishing/updating a game asset from an IDE. For this scenario we developed an Eclipse IDE plug-in. The asset developer opens the asset project in the Eclipse IDE; using the plugin the developer creates/updates the asset in RAGE Asset Repository within the IDE, providing credentials and needed metadata.

The fourth scenario: Asset consumers can search for a game asset using full text or advanced search, browse the repository, view assets metadata and download assets or artefacts for reuse.

Currently the repository is populated with the metadata of all currently developed Assets in RAGE project.

8. Usage of the RAGE Metadata editor

The Metadata editor has been used for creating and maintenance of metadata records of game assets designed and developed in the scope of the RAGE project. Here, we discuss the process of the creation of metadata records for two assets – the "Player-centric rule-and-pattern-based adaptation" asset and the "Real-time arousal detection using Galvanic Skin Response" (GSR) asset. The first asset is built as a pure software component searching for a pattern or a rule in development of given player's metrics (such as performance registered while solving a game task) and using its occurence for game adaptation purposes. In contrast with it, the second asset includes software artefacts plus a cheap custom hardware device for measuring the GSR signal from particular player, which is applied for inferring tonic and phasic arousal of the player. Documentation and artefacts of both the assets have been published in GitHub, therefore the second scenario from the previous section was followed. This scenario helps asset developers by harvesting artefacts and metadata from GitHub and publishing them in the RAGE Asset Repository. Next, we had to check records created automatically and to supply the additional metadata.

The RAGE Metadata editor provides a concise graphic interface for creation and update of an asset's metadata, which is structured into six sections (tabs) named Main, Classification, Status, Solution, and Usage (Figure 4), followed by an additional section showing the whole record. Each field (excluding names of taxonomy concepts, dates, and URL's) can be multiplied by additional fields of the same type providing the same content in languages other than English. Taxonomy concepts are to be defined within the Classification context of asset, where the editor provides embedded support of the Rage Taxonomy selector as shown in Figure 5. Taxonomy concepts checked and confirmed in the selector appear automatically below the taxonomy field, and vice versa. Multiple Classification contexts can be added and described by selection concepts from various taxonomies regarding game engine/genre/platform, asset status and type, etc.

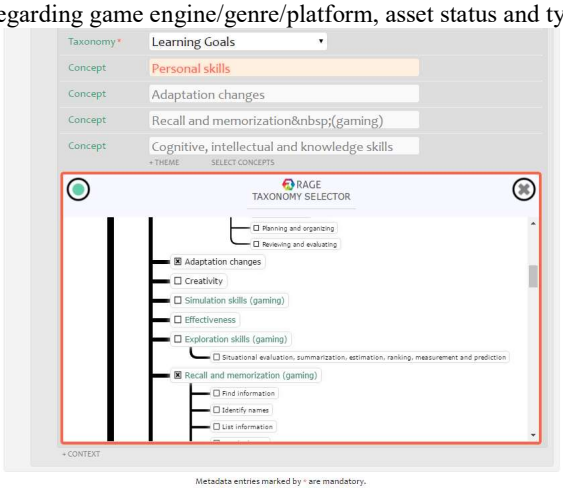

Figure 5. A partial view of the RAGE Metadata editor with Taxonomy Selector

Besides Classification, the tabs about Solution and Usage appear to be the heaviest sections of each asset metadata. Within the Solution tab, we specify metadata about one or more artefacts regarding an asset's requirements (e.g. use cases and other diagrams), design, implementation, and tests; while within the Usage tab we provide metadata about artefact(s) about installing, customisation and using

the asset such as integration and configuration tutorials and physical asset elements. In addition, for assets requiring a hardware component like the "Real-time arousal detection using GSR" asset, we have to specify all of the metadata of the device. For both the Solution and Usage sections users are able to add one or more custom metadata by specifying a key/value pair for each additional custom record.

9. Scenario evaluation

This section presents the conclusions based on our observations of real users. An evaluation of the usage scenarios was carried out by involving a group of 9 end users, viz. asset developers from the RAGE project. Preliminary findings support the relevance of the repository system. Comments about the first version of the repository and related client tools can be summarized as follows:

- Users can easily work with basic services such as searching, downloading or uploading assets to the repository.
- Users need more specific instructions how to populate the repository with metadata.
- The metadata editor improves the process of populating the repository for users.
- Users encounter problems to identify the source of the information related to some of the metadata fields, like keywords and others.
- There is a need to automate further the definition of metadata fields.

While the evaluation is preliminary and relatively informal initial response has been positive, and confirms the viability of this first step within the RAGE Project. Although the overall conclusion is that RAGE end-users accept the editor as a usable tool for entering their metadata, we did consider that external technology developers who may wish to upload their components to the RAGE repository, might be deterred by the complexity of the metadata and its documentation. In order to arrive at a sustainable ecosystem with a continuous influx of new technology assets from external parties, the metadata barriers should be as low as possible. Therefore, we designed and developed a workflow guidance wizard, which facilitates a stepwise process of metadata entry, without the requirement for extensive documentation. The wizard decomposes the process into 8 successive steps along the most relevant parts of the RAGE metadata scheme (which is kept hidden). The 8 steps of the wizard are:

1. About (eight introductory metadata fields requiring general information, e.g. title, description, etc.
2. Classification (six metadata fields requiring info about target platforms, programming language, applied computing keywords)
3. Status (five metadata fields with info about the software version, version notes, commit reference)
4. License (details about the licenses, conditions and potential restrictions)
5. Contacts (information about owners and creators)
6. Resources (six metadata fields with files or references to the software, documentation, tests, etc.)
7. Quality (five metadata fields with information about the asset's quality)
8. Submission (publishing the metadata in the repository)

A screenshot of step 1 of the wizard is presented in **Figure 6**.

Finally, we conducted a usability study of the asset creation wizard. We used three validated measurement instruments for evaluating the usability of the Asset Creation Wizard, namely, a System Usability Scale (SUS) [28], Form Usability Scale (FUS) [29,30], and the Usability Metric for User Experience (UMUX) [31].

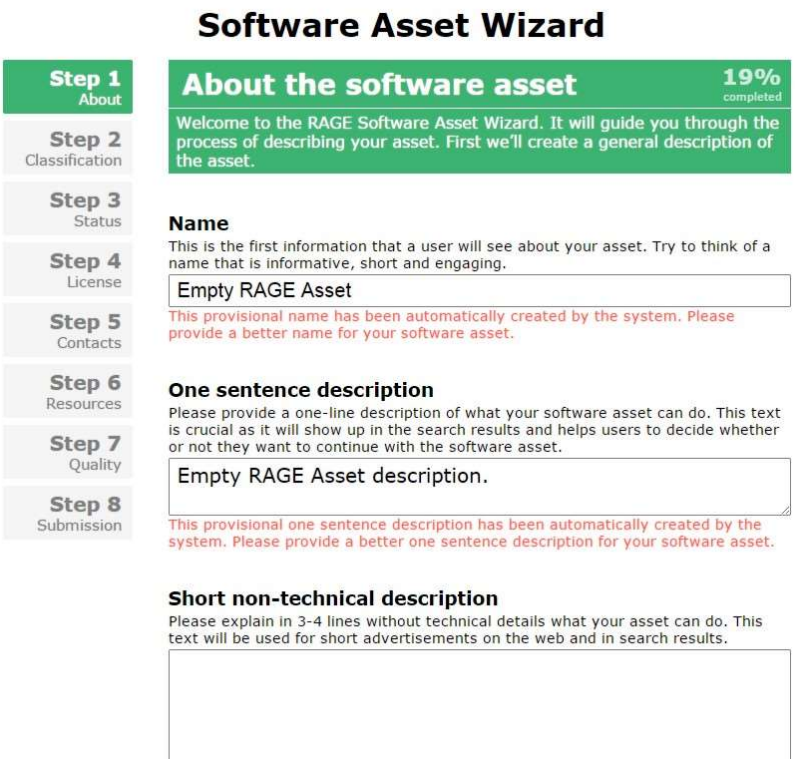

Figure 6. Screenshot of the asset creation wizard

SUS is a well-validated and reliable questionnaire applicable to a wide range of software systems. However, the questions in the SUS questionnaire are too generic to evaluate usability issues specific to online forms. For this reason, we employed the FUS questionnaire, which was specifically designed for evaluating the usability of online forms. While the SUS and the FUS questionnaires provide overall scores of usability, the UMUX questionnaire was developed to explicitly reflect the four separate components of usability as defined by ISO 9241- 11: efficiency, effectiveness, satisfaction, and overall usability. An optional open input field to comment the answer followed each question in the measurement instrument.

The SUS questionnaire consists of 10 questions. Responses are measured on a Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). The overall score per participant ranging from 0 to 100 with 50 as being neutral.

The FUS questionnaire consists of 10 questions. Validation of the questionnaire showed that question 7 provides little discriminatory value and information gain [29, 30]. Therefore, the question is excluded from our analysis. In our study, we used a 5-point Likert scale and removed the option for skipping. We normalized the FUS score to make it comparable with the SUS score. The overall score ranges from 0 to 100.

The UMUX questionnaire consists of four questions. The two questions regarding the satisfaction and overall usability overlap with two questions from the SUS questionnaire. Therefore, we have reused responses for the SUS questionnaire for evaluating these two usability components. We used a 5-point Likert scale for the purpose of consistency across questionnaires. The overall score is obtained by dividing the sum of four scores by 16 and then multiplying by 100.

In total, 15 asset developers participated in the study. They are members of the RAGE project from eight different organisations. Each one was instructed to use the wizard for submitting the metadata and artefacts of their game assets to the RAGE repository. All participants managed to successfully complete their submissions. Participants were asked to address and complete the questionnaire(s) after usage of the wizard.

The mean overall SUS score is 72.8 (SD=16.3, SE=4.2) where SD and SE are the standard deviation and the standard error, respectively. This overall usability evaluation is positive. All the participants with one exception positively evaluated the overall usability of the Wizard. The mean scores for all questions except one are positive. The question with the negative mean score (M=1.7,

SE=.3) is concerned with the frequency of using the Wizard. The negative score was anticipated as the Wizard is projected to be used infrequently.

The mean overall FUS score is 65.7 (SD=16.0, SE=4.1). Overall the usability evaluation of the Wizard is positive with room for improvement. Two participants negatively evaluated the overall usability of the Wizard. The correlation between the SUS and the FUS overall scores is significantly high ($r(13) = .67$, $p = .006$), which indicates that the FUS score is consistent with the benchmark score of the SUS.

The mean scores for nine questions are positive. One question has a negative mean score (Mean=1.9, SE=.3). The responses to this question indicates that the Wizard did not have sufficient feedback to users for resolving unexpected problems.

Participants provided 34 (M=3.4, SE=.6) and 42 (M=4.2, SE=.7) comments to their responses in the SUS and the FUS questionnaires, respectively. While overall usability scores are positive, the number of comments indicates that there may be some specific issues in the Wizard that should be further addressed. Finally, more comments in the FUS questionnaire indicate - as expected - that the FUS questionnaire did capture the issues that are specific to online forms.

The mean overall UMUX score is positive (M=72.5, SD=16.7, SE=4.3). The distribution of the overall scores also indicates positive evaluation with only one overall score being negative. The correlation of the UMUX scores with the SUS scores is significantly high ($r(13) = .88$, $p < .001$). The correlation of the UMUX scores with the FUS scores is significantly high as well ($r(13) = .74$, $p = .002$). The results indicate that the evaluation of all three usability components (effectiveness, satisfaction, efficiency) is positive. The Wizard is fit for purpose in managing metadata and artefacts. The participants reported positive efficiency indicating that both the asset and metadata management was fast and did not require substantial effort. Finally, the participants reported positive satisfaction towards using the Wizard.

Further fine-tuning of the wizard and its accompanying instructions will be addressed in the next version of the wizard software.

10. Workflow and Quality Assurance

The RAGE quality assurance processes are designed to encourage and support innovation and agility. These processes are designed to be lightweight and easily implementable to encourage wide engagement with the portal and expand use of the assets themselves.

Critical success factors for the software asset repository and the RAGE Eco-system is a rigorous and robust, approach to Quality Assurance (QA). It is the aim of the RAGE project team to strike an effective balance between; assuring the consistent integrity of the assets with easy to use, easily implementable application and software quality criteria.

To ensure the quality of assets within the repository and in particular those contributed to the portal by external Developers, who may not necessarily be familiar with the requirements and workflows established within the RAGE project, users are guided within the asset manager and metadata editor by an integrated widget, managed by system logic, to ensure version compatibility and consistent code, metadata and documentation. The Quality metrics include the following self-evaluated statements:

- **The metadata field (software version).**
  The RAGE project specifies the semantic versioning specification 2.0.0 (SemVer).
- **The overall source code quality.**
  The RAGE project requires confirmation from developers that any submitted code conforms to the specified RAGE architecture requirements. That the software uses consistent code and is well documented using defined documentation guidelines. The software is portable and avoids any platform specific calls. Also, a series of general code review checks are undertaken.
- **The software testing.**
  The software includes (unit) tests and has passed the required tests (and includes test reports). The software includes a set of dummy data for testing, and uses a Continuous Integration system. The software has passed (if applicable) load tests (and includes test reports) and if applicable in-game performance tests (and includes test reports)
- **The software manuals and documentation.**
  The software includes API and release documentation.
- The software includes the corresponding license this has been determined as the Apache 2.0 for original RAGE software assets. The software includes build and run instructions and

documents all its dependencies and platform requirements. The asset provides clear documentation referring to the originator and owner with contact details for support and enquiry.

- **Metadata.**
  That the software complies with RAGE metadata guidelines and schemas
- **The non-software artefacts.**
  Asset promotional video and video tutorials with quick-start guides.
- Installation and user manuals are provided.
- Software testing reports are included in the asset with proof cases and authoring tools.
- **An automated field of quality metrics.**
  Considering each of the other metrics above RAGE may define and publish an overall quality metric that reflects assets completeness and conformity.
- **Any other requirements** (This includes such things as usability, fitness for purpose, compatibility, packaging and integration.

Where practical and possible it is intended that the checks and balances defined in the Quality assurance processes detailed above will be automated ; mindful that a positive user experience is critical in supporting wider stakeholder engagement that and in underpinning the establishment of a sustainable Eco-system.

11. Conclusions and future work

In this paper, we presented a unique software architecture supporting the lifecycle of reusable software components for applied gaming. We have built using the best practices as described in the research literature, including RAS based asset repositories, metadata based educational repositories, and game asset stores. The software architecture plays a pivotal role within the RAGE Ecosystem, developed for the RAGE project and is considered of strategic significance for the applied gaming domain.

The repository as the content core system of the RAGE Ecosystem allows for flexible design and development of RAGE game assets and future search, packaging and exchange. The current architecture guarantees both scalability and durability. It also provides a high level of flexibility across different taxonomies and standards.

Future work is planned on improving the architecture by automating Quality Assurance services and asset development workflows, harvesting of assets from external systems and stores, adding social functions for making the work for developers more easy and natural, and for specific targeted support for the gaming community. A provisional launch of the repository integrated in the RAGE social platform is expected later in 2017.

The repository presented in this paper and the wide range of easy-to-use game technology components that it will expose are an important measure to counteract the fragmentation of the applied game industry.

The approach detailed holds the promise to help unite, interconnect, and harmonise the applied game industry in Europe and will help further in establishing it as an authentic sub set of the wider Games Industry.

Acknowledgements.

This work has been partially funded by the EC H2020 project RAGE (Realising an Applied Gaming Eco-System); http://www.rageproject.eu/; Grant agreement No 644187 and by national funds provided through Fundacao para a Ciencia e a Tecnologia (UID/CEC/50021/2013).

References

[1] García Sánchez, R., Baalsrud Hauge, J., Fiucci, G., Rudnianski, M., Oliveira, M., Kyvsgaard Hansen, P., Riedel, J., Brown, D., Padrón-Nápoles, C.L., Arambarri Basanez, J., Business Modelling and Implementation Report 2, GALA Network of Excellence, , Deliverable D4.10, 2013.

[2] Stewart, J., Bleumers, L., Van Looy, J., Mariën, I., All, A., Schurmans, D., Willaert, K., De Grove, F., Jacobs, A., Misuraca, G., The Potential of Digital Games for Empowerment and Social Inclusion of Groups at Risk of Social and Economic Exclusion: evidence and opportunity for policy. In Centeno C. (ed.) Joint Research Centre, European Commission, 2013
[3] "RAGE: Project Web site" http://www.rageproject.eu .
[4] Ackerman, L., Elder, P., Busch, C.V., Lopez-Mancisidor, A., Kimura, J., Balaji, N.A., Strategic reuse with asset-based development, IBM RedBooks, 2008
[5] Norfolk, D., Atego Asset Library, White Paper, Bloor research, June 2013
[6] Hong-min, R., Zhi-ying, Y., Jing-zhou, Z., "Design and Implementation of RAS-Based Open Source Software Repository", in Proceedings of the Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Vol.2, pp.219-223, 2009
[7] Moura, D. S., Software Profile RAS: estendendo a padronização do Reusable Asset Specification e construindo um repositório de ativos, Master's thesis, Univ. Federal do Rio Grande do Sul, Brasil, 2013
[8] Hilliar, G., Developing Cross-Platform Mobile Apps with HTML5 and Intel XDK, Dr. Dobb's Journal, November 2014, UBM plc.
[9] Böhm, T., Klas, C.-P., Hemmje, M., "Supporting Collaborative Information Seeking and Searching in Distributed Environments". In Proceedings of the LWA 2013 Conference, Bamberg, Germany, 2013
[10] Stefanov, K., Nikolov, R., Boytchev, P., Stefanova, E., Georgiev, A., Koychev, I., Nikolova, N., Grigorov, A., "Emerging Models and e-Infrastructures for Teacher Education", in Proceedings of the 2011 International Conference on Information Technology Based Higher Education and Training ITHET 2011, IEEE Catalog Number: CFP11578-CDR, ISBN: 978-1-4577-1671-3, 2011 https://doi.org/10.1109/ITHET.2011.6018688
[11] Carvalho, M. B., Bellotti, F., Berta, R., De Gloria, A., Gazzarata, G., Hu, J., Kickmeier-Rust, M., A case study on Service-Oriented Architecture for Serious Games, Entertainment Computing, Vol. 6, January 2015, pp. 1-10, http://dx.doi.org/10.1016/j.entcom.2014.11.001
[12] De Gloria, A., Bellotti, F., Berta, R., and Lavagnino, E., Serious Games for Education and Training, International Journal of Serious Games, Vol. 1, No. 1, 2014 https://doi.org/10.17083/ijsg.v1i1.11
[13] Vegt, W. van der, Westera, W., Nyamsuren, E., Georgiev, A. and Martínez Ortiz, I., RAGE Architecture for Reusable Serious Gaming Technology Components. International Journal of Computer Games Technology, vol. 2016, Article ID 5680526, 2016, http://dx.doi.org/10.1155/2016/5680526
[14] Dekkers, M., Asset Description Metadata Schema (ADMS). W3C Working Group (2013)
[15] Vegt, W. van der, Nyamsuren, E., Westera, W., Martinez Ortiz, I., "RAGE Reusable Game Software Components and their Integration into Serious Game Engines", in Kapitsaki G. & S. de Almeida E. (eds) Software reuse: Bridging the Social Awareness. ICSR 2016. LNCS Vol 9679, Springer, 2016
[16] Georgiev, A. Grigorov, B. Bontchev, P. Boytchev, K. Stefanov, K. Bahreini, E. Nyamsuren, W. van der Vegt, W. Westera, R. Prada, P. Hollins, P. Moreno, "The RAGE Software Asset Model and Metadata Model", Serious Games, 2nd Joint International Conference, JCSG 2016, Lecture Notes in Computer Science, Vol. 9894, pp. 191-203, Springer, 2016
[17] Martin, J., Rapid Application Development, Macmillan, 1991
[18] Hollins, P. Westera,W. Manero Iglesias, B., "Amplifying applied game development and uptake", In Proceedings of 9th European Conference on Game-Based Learning ECGBL 2015, pp. 234-241, Steinkjer, Norway, 2015
[19] Duval, E., Hodgins, W., Sutton, S., Weibel, S. L., Metadata principles and practicalities. D-lib Magazine, Vol 8, Nr 4, 2002, http://dx.doi.org/10.1045/april2002-weibel
[20] "Fedora 4.3 Documentation", https://wiki.duraspace.org/display/FEDORA43/
[21] Broekstra, J., Kampman, A., van Harmelen, F., "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema". First International Semantic Web Conference, Lecture Notes in Computer Science, pp 54--68, Springer, 2002
[22] Smiley, D., Pugh, E., Parisa, K., Mitchell, M., Apache Solr 4 Enterprise Search Server, Packt Publishing, 2014
[23] Lagoze, C., Van de Sompel, H., The Open Archives Initiative Protocol for Metadata Harvesting, 2015
[24] SPARQL 1.1: SPARQL 1.1 Overview, W3C Recommendation, 2013
[25] JSON-LD 1.0: A JSON-based Serialization for Linked Data, W3C Recommendation, 2014
[26] SKOS: Simple Knowledge Organization System Reference, W3C Recommendation, 2009

[27] "GitHub API: GitHub Developer Guide", 2016, https://developer.github.com/v3/
[28] Brooke, J., "SUS – A quick and dirty usability scale". In Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester and Ian L. McLelland (eds.) Usability Evaluation in Industry, Taylor and Francis, 1996, pp. 189-194.
[29] Aeberhard, A., FUS-Form Usability Scale, Development of a Usability Measuring Tool for Online Forms. Master's Thesis, University of Basel, Switzerland, 2011.
[30] Seckler, M., Heinz, S., Bargs-Avila, J.A., Opwis, K. & Tuch A.N., "Designing usable web forms: empirical evaluation of web form improvement guidelines". In Proceedings of the 32nd annual ACM Conference on Human Factors in Computing Systems, ACM, 2014, pp. 1275-1284. https://doi.org/10.1145/2556288.2557265
[31] Finstad, K., The usability metric for user experience. Interacting with Computers, Vol 22 (5), 2010, pp. 323-327. https://doi.org/10.1016/j.intcom.2010.04.004